UNITED STATES PATENT AND TRADEMARK OFFICE

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/809,247	03/25/2004	Alan Tchochiev	MSFT122122	7234

26389   7590   06/26/2008
CHRISTENSEN, O'CONNOR, JOHNSON, KINDNESS, PLLC
1420 FIFTH AVENUE
SUITE 2800
SEATTLE, WA 98101-2347

EXAMINER
WANG, BEN C

ART UNIT	PAPER NUMBER
2192	

MAIL DATE	DELIVERY MODE
06/26/2008	PAPER

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

		Application No.	Applicant(s)
Office Action Summary		10/809,247	TCHOCHIEV, ALAN
		Examiner	Art Unit
		BEN C. WANG	2192

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE <u>3</u> MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.
- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133).
  Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1) ☒ Responsive to communication(s) filed on <u>29 February 2008</u>.

2a) ☐ This action is **FINAL**. 2b) ☒ This action is non-final.

3) ☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4) ☒ Claim(s) <u>1-56</u> is/are pending in the application.

    4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5) ☐ Claim(s) _____ is/are allowed.

6) ☒ Claim(s) <u>1-56</u> is/are rejected.

7) ☐ Claim(s) _____ is/are objected to.

8) ☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9) ☐ The specification is objected to by the Examiner.

10) ☐ The drawing(s) filed on _____ is/are: a)☐ accepted or b)☐ objected to by the Examiner.

    Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

    Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11) ☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12) ☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

    a)☐ All b)☐ Some * c)☐ None of:

      1. ☐ Certified copies of the priority documents have been received.

      2. ☐ Certified copies of the priority documents have been received in Application No. _____.

      3. ☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

    * See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☒ Notice of References Cited (PTO-892)

2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)

3) ☐ Information Disclosure Statement(s) (PTO/SB/08) Paper No(s)/Mail Date _____.

4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____

5) ☐ Notice of Informal Patent Application

6) ☐ Other: _____.

DETAILED ACTION

1.   Applicant's amendment dated February 29, 2008, responding to the Office action

mailed October 30, 2007 provided in the rejection of claims 1-56, wherein claims 1, 12,

14, 53, and 55 were amended.

Claims 1-56 remain pending in the application and which have been fully

considered by the examiner.

Applicant's arguments with respect to claims rejection have been fully considered but

are moot in view of the new grounds of rejection – see Biernat, Sun, and Hibernate - arts made

of record, as applied hereto.

Claim Rejections – 35 USC § 103(a)

The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all

obviousness rejections set forth in this office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set
> forth in section 102 of this title, if the differences between the subject matter sought to be patented
> and the prior art are such that the subject matter as a whole would have been obvious at the time the
> invention was made to a person having ordinary skill in the art to which said subject matter pertains.
> Patentability shall not be negatived by the manner in which the invention was made.

2.   Claims 1-6 are rejected under 35 U.S.C. 103(a) as being unpatentable over Tim

Biernat, (*Persistent Object ID Service – Part1, IBM, June 2, 2003, pp. 1-6*) (hereinafter

'Biernat' - art made of record) in view of Sun (*Using the Timer Service, Sun*

Microsystems, December 4, 2003, pp. 1-9) (hereinafter 'Sun' - art made of record) and

further in view of Hibernate (*HIBERNATE – Relational Persistence for Idiomatic Java,*

hibernate, October 13, 2003, pp. 1-123) (hereinafter 'Hibernate' - art made of record)

3.   **As to claim 1** (Currently Amended), Biernat discloses a computer-readable

medium having a base generator class (e.g., P. 2, Fig. 1 – Initial class diagram design –

PoidGenerator; P. 4, Fig. 4 - Detailed sequence diagram, element - PoidGenerator)

stored thereon for use by developers to create generators to perform specific tasks, the

base generator class comprising:

- a base generator class constructor for initializing a generator (e.g., P. 3, Sec. of
  "Implementation" – PoidGenerator Initialize(), 1st Par. - ... some initialization will
  need to performed in the PoidGenerator ... which is instantiated in the
  initialization sequence, and loads the properties file ...);

- a generator properties class that provides incrementation capability, which allows
  the value of a generator property to vary during consecutive executions of a
  generator (e.g., Sec. of "Introduction", 2nd Par. - ... the ability to generate unique
  global identifiers for persistent objects ... as persistent object IDs (POIDs).
  POIDs provide a long lived object with a global identity that transcends space
  and time ...; Sec. of "Design", 2nd Par. - ... a singleton class named
  PoidGenerator, exposing a simple API, and responsible for managing blocks of
  POIDs);

- a status indicator (e.g., Sec. of "Introduction", 3rd Par. - ... employ standard J2EE
  ... Enterprise Java Beans (EJB) and Simple Object Access Protocol (SOAP) ...;
  Sec. of "Design", 2nd Par. - ... can provide clients an easy to use in-process

component, which will be responsible for service location and communication,

and POID block management …);

Further, Biernat discloses the ability to generate unique global identifiers for

persistent objects and using standard <u>J2EE</u> and Web technologies, including <u>EJBs</u> and

SOAP (e.g., Abstract) but does not explicitly disclose a schedule class.

However, in an analogous art of *Using the Timer Service*, Sun discloses a schedule

class (e.g., Abstract, 1st Par. - … can schedule a timed notification to occur at a specific

time, after a duration of time, or at timed intervals)

Therefore, it would have been obvious to one of ordinary skill in the art, at the time

the invention was made to combine the teachings of Sun into the Biernat's system to

further provide a schedule class in the Biernat's system.

The motivation is that it would further enhance the Biernat's system by taking,

advancing and/or incorporating Sun's system which offers significant advantages that

the timer service of the EJB container enable you to schedule timed notifications for all

types of enterprise beans as once suggested by Sun (e.g., Abstract, 1st Par.)

Furthermore, Biernat discloses the ability to generate unique global identifiers for

persistent objects (e.g., Abstract) and Sun discloses the timer service of the EJB

container enable you to schedule timed notifications for all types of enterprise beans

(e.g., Abstract, 1st Par.); but both do not explicitly disclose a logging class; and wherein

the logging class is used to verify the tasks performed by the generators.

However, in an analogous art of *Hibernate – Relational Persistence for Idiomatic

Java*, Hibernate discloses a logging class; and wherein the logging class is used to

verify the tasks performed by the generators (e.g., Sec. 2.7 – Logging, 1st Par. - … Hibernate <u>logs various events</u> using Apache commons-logging …; 2nd Par. - … a lot of work has been put into making the Hibernate log as detailed as possible … It is <u>an essential troubleshooting device</u>)

    Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Hibernate into the Biernat-Sun's system to further provide a logging class; and wherein the logging class is used to verify the tasks performed by the generators in the Biernat-Sun's system.

    The motivation is that it would further enhance the Biernat-Sun's system by taking, advancing and/or incorporating Hibernate's system which offers significant advantages that the as once suggested by Hibernate (e.g., Sec. 1.3 – JMX Integration, 1st Par. - … JMX is the J2EE standard for management of java components. Hibernate may be managed via a JMX standard MBean)

4.   **As to claim 2** (Previously Presented) (incorporating the rejection in claim 1), Biernat discloses the computer-readable medium wherein the generator properties class that provides incrementation capability includes a plurality of generator properties (e.g., Sec. of "Introduction", 2nd Par. - … <u>the ability to generate unique global identifiers for persistent objects</u> … <u>as persistent object IDs</u> (POIDs). POIDs provide <u>a long lived object with a global identity</u> that transcends space and time …; Sec. of "Design", 2nd Par. - … a singleton class named PoidGenerator, exposing a simple API, and responsible for managing blocks of POIDs)

5.   **As to claim 3** (Original) (incorporating the rejection in claim 2), Biernat discloses

the computer-readable medium wherein said plurality of generator properties includes:

- a value of a generator property (e.g., P. 2, Fig. 1 – Initial class diagram design,

  element - PoId);

- a plurality of incrementation settings (e.g., P. 2, Fig. 1 – Initial class diagram

  design, element - PoIdBlock); a default incrementor that changes the value of

  the generator property (e.g., P. 2, Fig. 1 – Initial class diagram design, element -

  PoIdGenerator – getPoId()); and

  Hibernate discloses a default validator that validates the value of the generator

property (e.g., Sec. 3.4 - Validatable)

6.   **As to claim 4** (Original) (incorporating the rejection in claim 1), Biernat discloses

the computer-readable medium wherein the status indicator includes a status user

interface (UI) for displaying the execution status of generators (e.g., Sec. of

"Introduction", 3rd Par. - … employ standard J2EE … Enterprise Java Beans (EJB) and

Simple Object Access Protocol (SOAP) …; Sec. of "Design", 2nd Par. - … can provide

clients an easy to use in-process component, which will be responsible for service

location and communication, and POID block management …)

7.   **As to claim 5** (Previously Presented) (incorporating the rejection in claim 1),

Biernat discloses the computer-readable medium wherein the schedule class

comprises:

- a start condition under which the execution of a generator may be started (e.g., P. 3, PoldGenerator initialize(), 1st Par. - ... which is instantiated in the initialization sequence, and loads the properties file ...);

- a recurrence condition under which the execution of a generator may recur;

- an end condition under which the execution of a generator stops (e.g., P. 4, Fig. 4 – Detailed sequence diagram, element PoldBlock – baseValue, blockSize, maxValue); and

- a dialog box that can be used to accept user input (e.g., P. 4, Fig. 4 – Detailed sequence diagram, element PoldBlock – baseValue, blockSize, maxValue)

8.   **As to claim 6** (Previously Presented) (incorporating the rejection in claim 1), Hibernate discloses the computer-readable medium wherein the logging class enables the recording of the execution process of a generator (e.g., Sec. 2.7 – Logging, 1st Par. - ... Hibernate logs various events using Apache commons-logging ...; 2nd Par. - ... a lot of work has been put into making the Hibernate log as detailed as possible ... It is an essential troubleshooting device)

9.   Claims 7-17, 19-26, and 28-29 are rejected under 35 U.S.C. 103(a) as being unpatentable over in view of Hibernate in view of Kaoru Yoneyama (Pat. No. US 6,985,892 B2) (hereinafter 'Yoneyama')

10.   **As to claim 7** (Previously Presented), Hibernate discloses a method of creating

a generator, wherein the generator performs a specific task, comprising:

- creating a new generator class that inherits a base generator class (e.g., Sec.

  4.1.4.1. Generator - ... all generators implement the interface

  net.sf.hibernate.id.IdentifierGenerator ... some applications may choose to

  provide their own specialized implementations);

- creating a public default constructor for the new generator class that overrides

  the base generator class constructor (e.g., Sec. 4.1.4.1. Generator - ... all

  generators implement the interface net.sf.hibernate.id.IdentifierGenerator ...

  some applications may choose to provide their own specialized

  implementations);

- implementing a function in the new generator class to perform the specific task

  (e.g., Sec. 4.1.4.1. Generator - ... some applications may choose to provide their

  own specialized implementations);

- verifying the task based on properties of the generator (e.g., Sec. 3.4 -

  Validatable);

Furthermore, Hibernate discloses various generators with incrementation capability

(Sec. 4.1.4.1. Generator; Sub-Sec. Increment) but does not explicitly disclose creating a

file.

However, in an art of *Method and Apparatus for Producing a File Name in an Image*

Manipulating System Having a Memory Device in Which a File Name and a Second

Train of Characters is Provided wherein a File Number is automatically Generated by

Incrementing a File Number Previously Assigned and Stored in Memory, Yoneyama

discloses creating a file (e.g., Abstract … <u>a file generator</u> … <u>increments</u> the last stored

file number and stores it in readiness for use <u>to identify the next file to be created</u> …)

    Therefore, it would have been obvious to one of ordinary skill in the art, at the time

the invention was made to combine the teachings of Yoneyama into the Hibernate's

system to further provide creating a file in the Hibernate's system.

    The motivation is that it would further enhance the Hibernate s system by taking,

advancing and/or incorporating Yoneyama's system which offers significant advantages

that when a plurality of systems of the same kind are installed, since any of the systems

is distinguished from another with the fourth character of a file name, the system can be

differentiated from one to another as once suggested by Yoneyama (e.g., Col. 12, Lines

57-67)

11.   **As to claim 8** (Original) (incorporating the rejection in claim 7), Hibernate

discloses the method wherein creating a public default constructor comprises:

-   initializing the base generator class constructor with the name and the

  description of the generator (e.g., Sec. 3.1.2 – Implement a default constructor);

  and

- defining the properties of the generator (e.g., Sec. 3.1.3 – Provide an identifier

  property)

12.   **As to claim 9** (Original) (incorporating the rejection in claim 8), Hibernate

discloses the method wherein defining properties for the generator comprises:

- (a) defining the name of a property (e.g., Sec. 3.1.3 – Provide an identifier

    property);

- (b) setting a default value for the property (e.g., Sec. 4.1.9 - property);

- (c) providing a description for the property (e.g., Sec. 4.1.9 - property);

- (d) specifying incrementation settings for the property (e.g., Sec. 4.1.4.1 –

    Generator, Sub-Sec. – increment);

- (e) creating a custom property incrementor, if applicable (e.g., Sec. 4.1.4.1 –

    Generator, 2nd Par. – ... to provide their own specialized implementations ...)

- (f) creating a custom property validator, if applicable (e.g., Sec. 3.4 - Validatable);

    and

- (g) repeating (a)-(f) for all properties of the generator.

13.   **As to claim 10** (Original) (incorporating the rejection in claim 17), Hibernate

discloses the method further comprising implementing a function to be executed before

each execution of a generator (e.g., Sec. 4.1.4.1. Generator - ... some applications may

choose to provide their own specialized implementations)

14.   **As to claim 11** (Original) (incorporating the rejection in claim 7), Hibernate

discloses the method further comprising implementing a function to be executed after

each execution of a generator (e.g., Sec. 4.1.4.1. Generator - ... some applications may
choose to provide their own specialized implementations)

15.   **As to claim 12** (Currently Amended), Hibernate discloses a method of using a
generator that performs a specific task, comprising:

- customizing settings of the generator (e.g., Sec. 4.1.4.1. Generator - ... all
  generators implement the interface net.sf.hibernate.id.IdentifierGenerator ...
  some applications may choose to provide their own specialized
  implementations), the settings including incrementation settings that specify how
  the value of a generator property may vary between generated objects (e.g., Sec.
  4.1.4.1 - generator, Sub-Sec. Increment);
- executing the generator with the customized settings (e.g., Sec. 4.1.4.1.
  Generator - ... some applications may choose to provide their own specialized
  implementations);
- verifying the task based on the settings of the generator (e.g., Sec. 3.4 -
  Validatable)

Furthermore, Hibernate discloses various generators with incrementation capability
(Sec. 4.1.4.1. Generator; Sub-Sec. Increment) but does not explicitly disclose such as
including at least creating a file, the generated object including a file.

However, in an art of *Method and Apparatus for Producing a File Name in an Image
Manipulating System Having a Memory Device in Which a File Name and a Second
Train of Characters is Provided wherein a File Number is automatically Generated by*

Incrementing a File Number Previously Assigned and Stored in Memory, Yoneyama

discloses such as including at least creating a file, the generated object including a file

(e.g., Abstract … a file generator … increments the last stored file number and stores it

in readiness for use to identify the next file to be created …)

   Therefore, it would have been obvious to one of ordinary skill in the art, at the time

the invention was made to combine the teachings of Yoneyama into the Hibernate's

system to further provide such as including at least creating a file, the generated object

including a file in the Hibernate's system.

   The motivation is that it would further enhance the Hibernate s system by taking,

advancing and/or incorporating Yoneyama's system which offers significant advantages

that when a plurality of systems of the same kind are installed, since any of the systems

is distinguished from another with the fourth character of a file name, the system can be

differentiated from one to another as once suggested by Yoneyama (e.g., Col. 12, Lines

57-67)

16.   **As to claim 13** (Previously Presented) (incorporating the rejection in claim 12),

Hibernate discloses the method customizing the settings of the generator, is

accomplished through a user interface (e.g., Sec. 1.3 – JMX Integration, 1st Par. - …

JMX is the J2EE standard for management of java components. Hibernate may be

managed via a JMX standard MBean)

17.   **As to claim 14** (Currently Amended) (incorporating the rejection in claim 13),
Hibernate discloses the method further comprising:

- starting an object generator user interface;

- selecting the generator; and

- customizing properties of the generator (e.g., Sec. 4.1.4.1 – Generator, 2nd Par. –
  ... to provide their own specialized implementations ...)

18.   **As to claim 15** (Previously Presented) (incorporating the rejection in claim 14),
Hibernate does not explicitly disclose the method selecting a generator further
comprising adding the generator from files containing one or more generators (e.g.,
Sec. 4.1.4.1. Generator - ... all generators implement the interface
net.sf.hibernate.id.IdentifierGenerator ... some applications may choose to provide their
own specialized implementations)

19.   **As to claim 16** (Original) (incorporating the rejection in claim 14), Hibernate
discloses the method further comprising loading the settings of a generator from a file
(e.g., Sec. 2.1 - Programmatic Configuration)

20.   **As to claim 17** (Previously Presented) (incorporating the rejection in claim 14),
Hibernate discloses the method wherein customizing the properties of the generator
comprises:

- (a) selecting one property (e.g., Sec. 4.1.9 - property);

- (b) specifying a value of the one property (e.g., Sec. 4.1.9 - property);

- (c) specifying incrementation settings of the one property (e.g., Sec. 4.1.4.1 –
  Generator, Sub-Sec. – increment); and

- (d) repeating (a)-(c) until there are no more properties to be customized.

21.   **As to claim 19** (Original) (incorporating the rejection in claim 14), Hibernate does
not explicitly disclose the method further comprising setting logging options for
executing the generator (e.g., Sec. 2.7 – Logging, 1st Par. - … Hibernate logs various
events using Apache commons-logging …; 2nd Par. - … a lot of work has been put into
making the Hibernate log as detailed as possible … It is an essential troubleshooting
device)

22.   **As to claim 20** (Original) (incorporating the rejection in claim 14), Hibernate
discloses the method further comprising saving the settings of the generator (e.g., Sec.
2.1 – Programmatic Configuration)

23.   **As to claim 21** (Previously Presented) (incorporating the rejection in claim 12),
Hibernate discloses the method customizing the settings of a generator, is
accomplished programmatically (e.g., Sec. 4.1.4.1. Generator - … all generators
implement the interface net.sf.hibernate.id.IdentifierGenerator ... some applications may
choose to provide their own specialized implementations)

24.   **As to claim 22** (Previously Presented) (incorporating the rejection in claim 21),

Hibernate discloses the method further comprising:

- creating a new instance of the generator (e.g., Sec. 4.1.4.1. Generator - ... some

  applications may choose <u>to provide their own specialized implementations);</u>

- setting a number of objects to be generated by the generator (e.g., Sec. 4.1.4.1 -

  generator, Sub-Sec. Increment); and

- customizing properties of the generator (e.g., Sec. 4.1.4.1. Generator - ... all

  generators implement the interface net.sf.hibernate.id.IdentifierGenerator ...

  some applications may choose to provide their own specialized implementations)

25.   **As to claim 23** (Previously Presented) (incorporating the rejection in claim 22),

Hibernate discloses wherein customizing properties of the generator comprises:

- (a) setting values of the properties (e.g., 4.1.9 – property); and

- (b) specifying incrementation settings of the properties (e.g., Sec. 4.1.4.1 -

  generator, Sub-Sec. Increment)

26.   **As to claim 24** (Original) (incorporating the rejection in claim 21), Hibernate

discloses the method further comprising:

- creating a new instance of the generator (e.g., Sec. 4.1.4.1. Generator - ... all

  generators implement the interface net.sf.hibernate.id.IdentifierGenerator ...

  some applications may choose to provide their own specialized

  implementations); and

- loading saved settings of the generator from a file (e.g., Sec. 17.2.1 – The config file)

27.   **As to claim 25** (Original) (incorporating the rejection in claim 21), Hibernate discloses The method further comprising:

- creating a new instance of the generator (e.g., Sec. 4.1.4.1. Generator - ... all generators implement the interface net.sf.hibernate.id.IdentifierGenerator ... some applications may choose to provide their own specialized implementations);
- loading saved settings of the generator from a file (e.g., Sec. 17.2.1 – The config file); and
- implementing a function to execute the generator asynchronously (e.g., Sec. 4.1.4.1. Generator - ... some applications may choose to provide their own specialized implementations)

28.   **As to claim 26** (Original) (incorporating the rejection in claim 21), Bowman-Amuah discloses further comprising:

- creating a new instance of the generator (e.g., Sec. 4.1.4.1. Generator - ... all generators implement the interface net.sf.hibernate.id.IdentifierGenerator ... some applications may choose to provide their own specialized implementations);

- loading saved settings of the generator from a file (e.g., Sec. 17.2.1 – The config file)

- displaying an object generation status UI; and

- adding the generator to the object generation status UI (e.g., Sec. 1.3 – JMX Integration, 1st Par. - … JMX is the J2EE standard for management of java components. Hibernate may be managed via a JMX standard MBean) and;

29.   **As to claim 28** (Original) (incorporating the rejection in claim 12), Hibernate discloses the method further comprising executing the generator through a user interface (e.g., Sec. 1.3 – JMX Integration, 1st Par. - … JMX is the J2EE standard for management of java components. Hibernate may be managed via a JMX standard MBean)

30.   **As to claim 29** (Original) (incorporating the rejection in claim 12), Hibernate discloses executing the generator through a user interface, but does not explicitly disclose the method further comprising executing the generator programmatically (e.g., Sec. 4.1.4.1. Generator - … all generators implement the interface net.sf.hibernate.id.IdentifierGenerator ... some applications may choose to provide their own specialized implementations)

31.   Claims 18 and 27 are rejected under 35 U.S.C. 103(a) as being unpatentable over in view of Hibernate in view of Yoneyama and further in view of Sun

32.   **As to claim 18** (Original) (incorporating the rejection in claim 14), Hibernate does not explicitly disclose the method further comprising setting a schedule for executing the generator.

Furthermore, Hibernate discloses various generators with incrementation capability (Sec. 4.1.4.1. Generator; Sub-Sec. Increment) but Hibernate and Yoneyama do not explicitly disclose method further comprising setting a schedule for executing the generator.

However, in an analogous art of *Using the Timer Service*, Sun discloses the method further comprising setting a schedule for executing the generator (e.g., Abstract, 1st Par. - ... can schedule a timed notification to occur at a specific time, after a duration of time, or at timed intervals)

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Sun into the Hibernate-Yoneyama's system to further provide a schedule class in the Hibernate-Yoneyama's system.

The motivation is that it would further enhance the Hibernate-Yoneyama's system by taking, advancing and/or incorporating Sun's system which offers significant advantages that the timer service of the EJB container enable you to schedule timed notifications for all types of enterprise beans as once suggested by Sun (e.g., Abstract, 1st Par.)

33.   **As to claim 27** (Original) (incorporating the rejection in claim 21), Hibernate

discloses the method further comprising:

- creating a new instance of the generator (e.g., Sec. 4.1.4.1. Generator - ... all

    generators implement the interface net.sf.hibernate.id.IdentifierGenerator ...

    some applications may choose to provide their own specialized

    implementations);

- loading saved settings of the generator from a file (e.g., . 17.2.1 – The config

    file); and

- displaying a logging dialog box that allows a user to specify logging options for

    executing the generator (e.g., Sec. 2.7 – Logging, 1st Par. - ... Hibernate <u>logs</u>

    <u>various events</u> using Apache commons-logging ...; 2nd Par. - ... a lot of work has

    been put into making the Hibernate log as detailed as possible ... It is <u>an</u>

    <u>essential troubleshooting device</u>)

Furthermore, Hibernate discloses various generators with incrementation capability

(Sec. 4.1.4.1. Generator; Sub-Sec. Increment) but Hibernate and Yoneyama do not

explicitly disclose displaying a schedule dialog box that allows a user to specify a

schedule for executing the generator.

However, in an analogous art of *Using the Timer Service*, Sun discloses

displaying a schedule dialog box that allows a user to specify a schedule for executing

the generator (e.g., Abstract, 1st Par. - ... can schedule a timed notification to occur at a

specific time, after a duration of time, or at timed intervals)

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Sun into the Hibernate-Yoneyama's system to further provide displaying a schedule dialog box that allows a user to specify a schedule for executing the generator in the Hibernate-Yoneyama's system.

The motivation is that it would further enhance the Hibernate-Yoneyama's system by taking, advancing and/or incorporating Sun's system which offers significant advantages that the timer service of the EJB container enable you to schedule timed notifications for all types of enterprise beans as once suggested by Sun (e.g., Abstract, 1st Par.)

34. Claims 30-56 are rejected under 35 U.S.C. 103(a) as being unpatentable over Biernat in view of Hibernate

35. **As to claim 30** (Previously Presented), Biernat discloses a method for object generation using a base generator class, comprising:

- creating a generator that performs a specific task (e.g., P. 2, Fig. 1 – Initial class diagram design – element PoIdGenerator; last Par. – request an instance of PoIdGenerator, then invoke getPoId() on the instance, passing …);

- customizing settings of the generator, that specify how the value of a generator property may vary between generated objects (e.g., Sec. of "Introduction", 2nd Par. - … the ability to generate unique global identifiers for persistent objects …

as persistent object IDs (POIDs). POIDs provide a long lived object with a global

identity that transcends space and time ...; Sec. of "Design", 2nd Par. - ... a

singleton class named PoIdGenerator, exposing a simple API, and responsible

for managing blocks of POIDs);

- executing the generator with the customized settings (e.g., Sec. of "Introduction",

   2nd Par. - ... example use cases include creating account numbers of bank

   accounts, and generating identifiers for new customers ...);

Further, Biernat discloses the ability to generate unique global identifiers for

persistent objects and using standard J2EE and Web technologies, including EJBs and

SOAP (e.g., Abstract) but does not explicitly disclose verifying the task based on the

settings of the generator;

However, in an analogous art of *Hibernate – Relational Persistence for Idiomatic

Java*, Hibernate discloses verifying the task based on the settings of the generator (e.g.,

Sec. 3.4 - Validatable)

Therefore, it would have been obvious to one of ordinary skill in the art, at the time

the invention was made to combine the teachings of Hibernate into the Biernat's system

to further provide verifying the task based on the settings of the generator in the

Biernat's system.

The motivation is that it would further enhance the Biernat's system by taking,

advancing and/or incorporating Hibernate's system which offers significant advantages

that the as once suggested by Hibernate (e.g., Sec. 1.3 – JMX Integration, 1st Par. - ...

JMX is the J2EE standard for management of java components. Hibernate may be managed via a JMX standard MBean)

36.   **As to claim 31** (Original) (incorporating the rejection in claim 30), Hibernate discloses the method wherein creating a generator that performs a specific task comprises:

- creating a new generator class that inherits the base generator class (e.g., Sec. 4.1.4.1. Generator - ... all generators implement the interface net.sf.hibernate.id.IdentifierGenerator ... some applications may choose to provide their own specialized implementations);

- creating a public default constructor for the new generator class that overrides the base generator class constructor (e.g., Sec. 4.1.4.1. Generator - ... all generators implement the interface net.sf.hibernate.id.IdentifierGenerator ... some applications may choose to provide their own specialized implementations); and

- implementing a function in the new generator class to perform a specific task (e.g., Sec. 4.1.4.1. Generator - ... some applications may choose to provide their own specialized implementations).

37.   **As to claim 32** (Original) (incorporating the rejection in claim 31), please refer to claim **8** as set forth accordingly.

38.   **As to claim 33** (Previously Presented) (incorporating the rejection in claim 32), please refer to claim **9** as set forth accordingly.

39.   **As to claim 34** (Previously Presented) (incorporating the rejection in claim 31), please refer to claim **10** as set forth accordingly.

40.   **As to claim 35** (Previously Presented) (incorporating the rejection in claim 31), please refer to claim **11** as set forth accordingly.

41.   **As to claim 36** (Original) (incorporating the rejection in claim 30), please refer to claim **13** as set forth accordingly.

42.   **As to claim 37** (Previously Presented) (incorporating the rejection in claim 36), please refer to claim **14** as set forth accordingly.

43.   **As to claim 38** (Previously Presented) (incorporating the rejection in claim 37), please refer to claim **15** as set forth accordingly.

44.   **As to claim 39** (Original) (incorporating the rejection in claim 37), please refer to claim **16** as set forth accordingly.

45.   **As to claim 40** (Previously Presented) (incorporating the rejection in claim 37),

please refer to claim **17** as set forth accordingly.

46.   **As to claim 41** (Original) (incorporating the rejection in claim 37), please refer to

claim **18** as set forth accordingly.

47.   **As to claim 42** (Original) (incorporating the rejection in claim 37), please refer to

claim **19** as set forth accordingly.

48.   **As to claim 43** (Original) (incorporating the rejection in claim 37), please refer to

claim **20** as set forth accordingly.

49.   **As to claim 44** (Previously Presented) (incorporating the rejection in claim 30),

please refer to claim **21** as set forth accordingly.

50.   **As to claim 45** (Previously Presented) (incorporating the rejection in claim 44),

please refer to claim **22** as set forth accordingly.

51.   **As to claim 46** (Previously Presented) (incorporating the rejection in claim 45),

please refer to claim **23** as set forth accordingly.

52.   **As to claim 47** (Original) (incorporating the rejection in claim 44), please refer to

claim **24** as set forth accordingly.

53.   **As to claim 48** (Original) (incorporating the rejection in claim 44), please refer to

claim **25** as set forth accordingly.

54.   **As to claim 49** (Original) (incorporating the rejection in claim 44), please refer to

claim **26** as set forth accordingly.

55.   **As to claim 50** (Original) (incorporating the rejection in claim 44), please refer to

claim **27** as set forth accordingly.

56.   **As to claim 51** (Original) (incorporating the rejection in claim 30), please refer to

claim **28** as set forth accordingly.

57.   **As to claim 52** (Original) (incorporating the rejection in claim 30), please refer to

claim **29** as set forth accordingly.

58.   **As to claim 53** (Currently Amended), Biernat discloses A method of varying a

value of a property associated with a task, during consecutive executions of the task

performed by a generator created to perform the task, comprising:

- allowing the value of the property to vary during consecutive executions of the task (e.g., Sec. of "Introduction", 2nd Par. - ... <u>the ability to generate unique global identifiers for persistent objects</u> ... <u>as persistent object IDs</u> (POIDs). POIDs provide <u>a long lived object with a global identity</u> that transcends space and time ...; Sec. of "Design", 2nd Par. - ... a singleton class named PoldGenerator, exposing a simple API, and responsible for managing blocks of POIDs);

- creating settings associated with the property that control how the value may vary during consecutive executions of the task (e.g., P. 3, Sec. of "Implementation" – PoldGenerator Initialize(), 1st Par. - ... <u>some initialization</u> will need to performed in the PoldGenerator ... <u>which is instantiated in the initialization sequence</u>, and <u>loads the properties file</u> ...; P. 4, Fig. 4 – Detailed sequence diagram, elements – PropertyManager, PoldBlock);

- allowing a user executing the task to customize the settings according to user preference (e.g., P. 4, Fig. 4 – Detailed sequence diagram, elements – PropertyManager, PoldBlock)

Further, Biernat discloses the ability to generate unique global identifiers for persistent objects and using standard J2EE and Web technologies, including EJBs and SOAP (e.g., Abstract) but does not explicitly disclose verifying the task based on the settings.

However, in an analogous art of *Hibernate – Relational Persistence for Idiomatic Java*, Hibernate discloses verifying the task based on the settings (e.g., Sec. 3.4 - Validatable)

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Hibernate into the Biernat's system to further provide verifying the task based on the settings in the Biernat's system.

The motivation is that it would further enhance the Biernat's system by taking, advancing and/or incorporating Hibernate's system which offers significant advantages that the as once suggested by Hibernate (e.g., Sec. 1.3 – JMX Integration, 1st Par. - ... JMX is the J2EE standard for management of java components. Hibernate may be managed via a JMX standard MBean)

59.   **As to claim 54** (Original) (incorporating the rejection in claim 53), Biernat discloses the method wherein the step of allowing the value of the property to vary during consecutive executions of the task further comprises:

- implementing a function that increments a property value according to the settings associated with the property that control how the value may vary during consecutive executions of the task (e.g., Sec. of "Introduction", 2nd Par. - ... the ability to generate unique global identifiers for persistent objects ... as persistent object IDs (POIDs). POIDs provide a long lived object with a global identity that transcends space and time ...; Sec. of "Design", 2nd Par. - ... a singleton class named PoldGenerator, exposing a simple API, and responsible for managing blocks of POIDs)

60.   **As to claim 55** (Currently Amended), Biernat discloses a computer-readable medium containing computer- executable instructions for a method of varying a value of

a property associated with a task, during consecutive executions of the task performed

by a generator created to perform the task, the method comprising:

- allowing the value of the property to vary during consecutive executions of the

    task (e.g., Sec. of "Introduction", 2nd Par. - ... the ability to generate unique global

    identifiers for persistent objects ... as persistent object IDs (POIDs). POIDs

    provide a long lived object with a global identity that transcends space and time

    ...; Sec. of "Design", 2nd Par. - ... a singleton class named PoidGenerator,

    exposing a simple API, and responsible for managing blocks of POIDs);

- creating settings associated with the property that control how the value may

    vary during consecutive executions of the task (e.g., P. 3, Sec. of

    "Implementation" – PoidGenerator Initialize(), 1st Par. - ... some initialization will

    need to performed in the PoidGenerator ... which is instantiated in the

    initialization sequence, and loads the properties file ...; P. 4, Fig. 4 – Detailed

    sequence diagram, elements – PropertyManager, PoidBlock);

- allowing a user executing the task to customize the settings according to user

    preference (e.g., P. 4, Fig. 4 – Detailed sequence diagram, elements –

    PropertyManager, PoidBlock);

Further, Biernat discloses the ability to generate unique global identifiers for

persistent objects and using standard J2EE and Web technologies, including EJBs and

SOAP (e.g., Abstract) but does not explicitly disclose verifying the task based on the

settings.

However, in an analogous art of *Hibernate – Relational Persistence for Idiomatic Java*, Hibernate discloses verifying the task based on the settings (e.g., Sec. 3.4 - Validatable)

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Hibernate into the Biernat's system to further provide verifying the task based on the settings in the Biernat's system.

The motivation is that it would further enhance the Biernat's system by taking, advancing and/or incorporating Hibernate's system which offers significant advantages that the as once suggested by Hibernate (e.g., Sec. 1.3 – JMX Integration, 1st Par. - ... JMX is the J2EE standard for management of java components. Hibernate may be managed via a JMX standard MBean)

61.   **As to claim 56** (Original) (incorporating the rejection in claim 55), Biernat discloses the computer-readable medium wherein the step of allowing the value of the property to vary during consecutive executions of the task further comprises:

- implementing a function that increments a property value according to the settings associated with the property that control how the value may vary during consecutive executions of the task (e.g., Sec. of "Introduction", 2nd Par. - ... the ability to generate unique global identifiers for persistent objects ... as persistent object IDs (POIDs). POIDs provide a long lived object with a global identity that transcends space and time ...; Sec. of "Design", 2nd Par. - ... a singleton class

named PoIdGenerator, exposing a simple API, and responsible for managing

blocks of POIDs)

Conclusion

62.   Any inquiry concerning this communication or earlier communications from the

examiner should be directed to Ben C. Wang whose telephone number is 571-270-

1240. The examiner can normally be reached on Monday - Friday, 8:00 a.m. - 5:00

p.m., EST.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's

supervisor, Tuan Q. Dam can be reached on 571-272-3695. The fax phone number for

the organization where this application or proceeding is assigned is 571-273-8300.

Information regarding the status of an application may be obtained from the

Patent Application Information Retrieval (PAIR) system. Status information for

published applications may be obtained from either Private PAIR or Public PAIR.

Status information for unpublished applications is available through Private PAIR only.

For more information about the PAIR system, see http://pair-direct.uspto.gov. Should

you have questions on access to the Private PAIR system, contact the Electronic

Business Center (EBC) at 866-217-9197 (toll-free). If you would like assistance from a

USPTO Customer Service Representative or access to the automated information

system, call 800-786-9199 (IN USA OR CANADA) or 571-272-1000.

/Ben C Wang/
Examiner, Art Unit 2192
June 19, 2008

/Tuan Q. Dam/
Supervisory Patent Examiner, Art Unit 2192